IBM federated database technology

Eileen Lin  
Laura Haas  
November 13, 2012  
(First published March 01, 2002)

In a large modern enterprise, information is almost inevitably distributed among several database management systems. Despite considerable attention from the research community, relatively few commercial systems have attempted to address this issue. This article describes the technology that enables clients of IBM's federated database engine to access and integrate the data and specialized computational capabilities of a wide range of relational and non-relational data sources.

Introduction

In a large modern enterprise, it is almost inevitable that different portions of the organization will use different database management systems to store and search their critical data. Competition, evolving technology, mergers, acquisitions, geographic distribution, and the inevitable decentralization of growth all contribute to this diversity. Yet it is only by combining the information from these systems that the enterprise can realize the full value of the data they contain.

For example, in the finance industry, mergers are an almost commonplace occurrence. The newly created entity inherits the data stores of the original institutions. Many of those stores will be relational database management systems, but often from different manufacturers; for instance, one company may have used primarily Sybase, and another Informix. They may both have one or more document management systems — such as Documentum or IBM Content Manager — for storing text documents such as copies of loans, and so on. Each may have applications that compute important information (for example, the risk of a loan to a given customer), or mine for information about customers' buying patterns.

After the merger, they need to be able to access all customer information from both sets of stores, analyze their new portfolios using existing and new applications, and, in general, use the combined resources of both institutions through a common interface. They need to be able to identify common customers and consolidate their accounts, although the different companies may have referred to their customers using totally different identifying keys. Federation technologies can significantly ease the pain in these situations by providing a unified interface to diverse data.

IBM has made significant investments in federation technologies that have resulted in market leading capabilities across the Information Management product portfolio. Today, federation capabilities enable unified access to any digital information, in any format — structured and
unstructured, in any information store. Federation capabilities are available today through a variety of IBM products including InfoSphere® Federation Server, DB2® for Linux, UNIX, and Windows, InfoSphere Warehouse, and IBM Enterprise Information Portal (EIP). This set of federation technologies continues to be enhanced and our customers’ investments in all of these products continue to deliver real business value.

This article focuses specifically on advanced database federation capabilities, implemented through a technology sometimes referred to by the code name "Garlic", which represent the next generation of information federation enhancements from IBM software. These enhancements will enable clients to access and integrate the data and specialized computational capabilities of a wide range of relational and nonrelational data sources. The Garlic technology will be incorporated into all IBM software offerings that provide federation technology over time. Customers may rest assured that not only will their investments in existing products be protected, but also that in the future, no matter which product is selected, they will be able to leverage the advanced capabilities described here.

IBM's federated database systems offer powerful facilities for combining information from multiple data sources. Built on best-of-breed technology from an earlier product, DB2 DataJoiner [3] (which evolved to be InfoSphere Federation Server), and enhanced with leading-edge features for extensibility and performance from the Garlic research project[2], IBM's federated database capabilities are unique in the industry. DB2 DataJoiner introduced the concept of a virtual database, created by federating together multiple heterogeneous relational data sources. Users of DB2 DataJoiner could pose arbitrary queries over data stored anywhere in the federated system, without worrying about the data's location, the SQL dialect of the actual data stores, or the capabilities of those stores. Instead, users had the full capabilities of DB2 against any data in the federation. The Garlic project demonstrated the feasibility of extending this idea to build a federated database system that effectively exploits the query capabilities of diverse, possibly non-relational data sources. In both of these systems, as in today's DB2, a middleware query processor develops optimized execution plans and compensates for any functionality that the data sources may lack.

In this article, we describe the key characteristics of IBM's federated technology: transparency, heterogeneity, a high degree of function, autonomy for the underlying federated sources, extensibility, openness, and optimized performance. We then "roll back the covers" to show how IBM's database federation capabilities work. We illustrate how the federated capabilities can be used in a variety of scenarios, and conclude with some directions for the future.

**Characteristics of the IBM federated solution**

**Transparency**

If a federated system is transparent, it masks from the user the differences, idiosyncracies, and implementations of the underlying data sources. Ideally, it makes the set of federated sources look to the user like a single system. The user should not need to be aware of where the data is stored (location transparency), what language or programming interface is supported by the data source (invocation transparency), if SQL is used, what dialect of SQL the source supports...
(dialect transparency), how the data is physically stored, or whether it is partitioned and/or replicated (physical data independence, fragmentation and replication transparency), or what networking protocols are used (network transparency). The user should see a single uniform interface, complete with a single set of error codes (error code transparency). IBM provides all these features, allowing applications to be written as if all the data were in a single database, although, in fact, the data may be stored in a heterogeneous collection of data sources.

**Heterogeneity**

Heterogeneity is the degree of differentiation in the various data sources. Sources can differ in many ways. They may run on different hardware, use different network protocols, and have different software to manage their data stores. They may have different query languages, different query capabilities, and even different data models. They may handle errors differently, or provide different transaction semantics. They may be as much alike as two Oracle instances, one running Oracle 11g, and the other Oracle 9i, with the same or different schemas. Or they may be as diverse as a high-powered relational database, a simple, structured flat file, a web site that takes queries in the form of URLs and spits back semi-structured XML according to some DTD, a Web service, and an application that responds to a particular set of function calls. IBM's federated database can accommodate all of these differences, encompassing systems such as these in a seamless, transparent federation.

**A high degree of function**

IBM's federated capability provides users with the best of both worlds: all the function of its rich, standard-compliant DB2 SQL capability against all the data in the federation, as well as all the function of the underlying data sources. DB2's SQL includes support for many complex query features, including inner and outer joins, nested sub-queries and table expressions, recursion, user-defined functions, aggregation, statistical analyses, automatic summary tables, and others too numerous to mention. Many data sources may not provide all of these features. However, users still get the full power of DB2 SQL on these sources' data, because of function compensation. Function compensation means that if a data source cannot do a particular query function, the federated database retrieves the necessary data and applies the function itself. For example, a file system typically cannot do arbitrary sorts. However, users can still request that data from that source (for example, some subset of a file) be retrieved in some order, or ask that duplicates be eliminated from that data. The federated database will simply retrieve the relevant data, and do the sort itself.

While many sources do not provide all the function of DB2 SQL, it is also true that many sources have specialized functionality that the IBM federated database lacks. For example, document management systems often have scoring functions that let them estimate the relevancy of retrieved documents to a user's search. In the financial industry, time-series data is especially important, and systems exist that can compare, plot, analyze, and subset time-series data in specialized ways. In the pharmaceutical industry, new drugs are based on existing compounds with particular properties. Special-purpose systems can compare chemical structures, or simulate the binding of two molecules. While such functions could be implemented directly, it is often more efficient and cost-effective to exploit the functionality that already exists in data sources.
and application systems. IBM allows the user to identify functions of interest from the federated sources, and then to use them in queries, so that no function of a source need be lost to the user of the federated system.

**Extensibility and openness of the federation**

All systems need to evolve over time. In a federated system, new sources may be needed to meet the changing needs of the users' business. IBM makes it easy to add new sources. The federated database engine accesses sources via a software component know as a wrapper. Accessing a new type of data source is done by acquiring or creating a wrapper for that source. The wrapper architecture enables the creation of new wrappers. Once a wrapper exists, simple data definition (DDL) statements allow sources to be dynamically added to the federation without stopping ongoing queries or transactions.

Any data source can be wrapped. IBM supports the ANSI SQL/MED standard [1] (MED stands for Management of External Data). This standard documents the protocols used by a federated server to communicate with external data sources. Any wrapper written to the SQL/MED interface can be used with IBM's federated database. Thus wrappers can be written by third parties as well as IBM, and used in conjunction with IBM's federated database.

**Autonomy for data sources**

Typically a data source has existing applications and users. It is important, therefore, that the operation of the source is not affected when it is brought into a federation. IBM's federated database does not disturb the local operation of an existing data source. Existing applications will run unchanged, data is neither moved nor modified, interfaces remain the same. The way the data source processes requests for data is not affected by the execution of global queries against the federated system, though those global queries may touch many different data sources. Likewise, there is no impact on the consistency of the local system when a data source enters or leaves a federation. The sole exception is during federated two phase commit processing for sources that participate. Data sources involved in the same unit of work will need to participate in commit processing and can be requested to roll back the associated changes if necessary.

Unlike other products, our wrapper architecture does not require any software to be installed on the machine that hosts the data source. We communicate with the data source by means if a client server architecture, using the source's normal client. In this way, IBM's federated data source looks like just another application to the source.

**Optimized performance**

The optimizer is the component of a relational database management system that determines the best way to execute each query. Relational queries are non-procedural and there are typically several different implementations of each relational operator and many possible orderings of operators to choose from in executing a query. While some optimizers use heuristic rules to choose an execution strategy, IBM's federated database considers the various possible strategies, modeling the likely cost of each, and choosing the one with the least cost. (Typically, cost is measured in terms of system resources consumed).
In a federated system, the optimizer must decide whether the different operations involved in a query should be done by the federated server or by the source where the data is stored. It must also determine the order of the operations, and what implementations to use to do local portions of the query. To make these decisions, the optimizer must have some way of knowing what each data source can do, and how much it costs. For example, if the data source is a file, it would not make sense to assume it was smart, and ask it to perform a sort or to apply some function. On the other hand, if the source is a relational database system capable of applying predicates and doing joins, it might be a good idea to take advantage of its power if it will reduce the amount of data that needs to be brought back to the federated engine. This will typically depend on the details of the individual query. The IBM optimizer works with the wrappers for the different sources involved in a query to evaluate the possibilities. Often the difference between a good and a bad decision on the execution strategy is several orders of magnitude in performance. IBM's federated database is unique in the industry in its ability to work with wrappers to model the costs of federated queries over diverse sources. As a result, users can expect the best performance possible from their federated system.

To further enhance performance, each wrapper implementation takes advantage of the operational knobs provided by each data source using the source's native API. For example, blocking multiple result rows into one message (a.k.a. block fetch) is a common performance knob. The query compiler will communicate with the wrapper to indicate which query fragments can utilize block fetch and thus achieve the maximal performance at runtime without loss of query semantics.

How IBM's federated capabilities work

Architecture

IBM's federated database architecture is shown in Figure 1. Applications can use any supported interface (including ODBC, JDBC, or a Web service client) to interact with the federated server. The federated server communicates with the data sources by means of software modules called wrappers.

Figure 1. Architecture of an IBM federated system
Configuring a federated system

A federated system is created by installing the federated engine and then configuring it to talk to the data sources. There are several steps to add a new data source to a federated system. First, a wrapper for the source must be installed, and IBM's federated database must then be told where to find this wrapper. This is done by means of a CREATE WRAPPER statement. If multiple sources of the same type are desired, only one wrapper is needed. For example, even if the federated system will include five Oracle database instances, possibly on different machines, only one Oracle wrapper is needed, and hence, only one CREATE WRAPPER statement will be required. However, each separate source must also be identified to the system. This is done via a CREATE SERVER statement. If there are five Oracle database instances, five CREATE SERVER statements must be issued.

For example, suppose there is a wrapper for accessing web sites, and a specific site from which the user wants to access data. The federated database can be informed about the wrapper via the following statement:

```
CREATE WRAPPER web_wrapper
LIBRARY "/u/haas/wrappers/libweb.a"
```

This statement basically tells the federated database where to find the code for the web_wrapper. Next, the federated database can be told about the actual web site to be used, by identifying it as a server associated with the web_wrapper.

```
CREATE SERVER weather_server
WRAPPER web_wrapper OPTIONS (URL 'http://www.weatherforecast.com')
```

The OPTIONS clause allows the basic CREATE SERVER statement to be tailored with information that the wrapper will need to access instances of this data source type.

After the wrapper and server have been defined, the data at the remote source must be described in terms of the data model of the federated middleware. Since the federated database described here supports an object-relational data model, each collection of data from an external source must be described to the federated engine as a table with columns of appropriate types. A collection of external data modeled as a table is called a nickname, and its table name and column names are used in the SQL submitted to the federation by applications. Nicknames are identified via a CREATE NICKNAME statement. The following statement sets up a nickname for a collection of information about the weather, and identifies the "columns" that can be used in a query.

```
CREATE NICKNAME weather
(zone integer, climate varchar(10), yearly_rainfall float)
SERVER weather_server OPTIONS (QUERY_METHOD 'GET')
```

The "OPTIONS" clause is again a way to pass information that the wrapper needs, this time in order to process queries against the nickname.

In addition to storing data, many data sources also have the capability to perform specialized searches or other computations. Such capabilities can be represented in SQL as userdefined...
functions. For example, the user might like to do a temperature forecast based on location and date, to identify customers for an air conditioning sale.

```sql
SELECT c.Name, c.Address
FROM customers c, stores s, weather w
WHERE temp_forecast(w.zone, :DATE) >= 85 AND c.ShopsAt = s.id and s.location = w.zone
```

Here, the function temp_forecast is used to represent the data source's ability to do a temperature forecast for the nickname weather. We refer to user-defined functions that are implemented by an external data source as **mapped** functions. Mapped functions are, once again, identified to the federated system via DDL statements. A CREATE FUNCTION statement tells the federated database that this is a function that can appear in a SELECT statement.

```sql
CREATE FUNCTION temp_forecast (integer, date) RETURNS float AS TEMPLATE
DETERMINISTIC NO EXTERNAL ACTION
```

The AS TEMPLATE clause tells the federated database that there is no local implementation of the function. Next, a CREATE FUNCTION MAPPING statement tells the federated database what server can evaluate the function. Several function mappings may be created for the same function. For our example, the following statement accomplishes the mapping:

```sql
CREATE FUNCTION MAPPING tf1 for temp_forecast SERVER weather_server
```

The above DDL statements produce metadata describing the information about nicknames and the signatures of mapped functions. This metadata is used by the federated query processing engine and is stored in the global catalogs of the federated database.

### Query processing

After the federated system is configured, an application can submit a query written in SQL to a federated server. The federated server optimizes the query, developing an execution plan in which the query has been decomposed into fragments that can be executed at individual data sources. As mentioned above, many decompositions of the query are possible, and the optimizer chooses among alternatives on the basis of minimum estimated total resource consumption. Once a plan has been selected, the federated database drives the execution, invoking the wrappers to execute the fragments assigned to them. To execute a fragment, the wrapper performs whatever data source operations are needed to carry it out, perhaps a series of function calls or a query submitted to the data source in its native query language. The resulting streams of data are returned to the federated server, which combines them, performs any additional processing that could not be accomplished by a data source, and returns the final result to the application.

At the heart of IBM's approach to federated query processing is the manner in which the federated server's optimizer and the wrappers together arrive at a plan for executing the query [4]. The optimizer is responsible for exploring the space of possible query plans. Dynamic programming is the default method used in join enumeration, with the optimizer first generating plans for single-table accesses, then for two-way joins, etc. At each level, the optimizer considers various join orders and join methods, and if all the tables are located at a common data source, it considers
performing the join either at the data source or at the federated server. This process is illustrated in Figure 2.

**Figure 2. Query planning for joins**

1. The IBM federated database submits candidate query fragments called "requests" to a wrapper if the query fragments apply to a single source.
2. When a non-relational wrapper receives a request, it determines what portion, if any, of the corresponding query fragment can be performed by the data source.
3. The wrapper returns a reply that describes the accepted portion of the fragment. The reply also includes an estimate of the number of rows that will be produced, an estimate of the total execution time, and a wrapper plan: an encapsulated representation of everything the wrapper will need to know to execute the accepted portion of the fragment.
4. The federated database optimizer incorporates the reply into a global plan, introducing additional operators as necessary to compensate for portions of fragments that were not accepted by a wrapper. The cost and cardinality information from the replies is used to estimate the total cost of the plan, and the plan with minimum total cost is selected from among all the candidates. When a plan is selected, it need not be executed immediately; it can be stored in the database catalogs and subsequently used one or more times to execute the query. Even if a plan is used immediately, it need not be executed in the same process in which it was created, as illustrated in Figure 3.

**Figure 3. Compilation and runtime for non-relational sources**

For example, consider a web site that exports information about stocks, including the exchange, opening and closing prices and the volumes traded. The web site can be searched by means of a form, and, as is often the case, the form allows the values of some but not all of the attributes to be constrained. Consider the following query:

```
SELECT TickerSymbol, StockName
FROM Stocks
WHERE Exchange='NYSE' AND Closing - Opening > 5
```

Since this is a singletable query, there is no need to consider join methods or orders, and only one fragment, consisting of the entire query, will be submitted as a request to the wrapper for the web data source. If the form allows the user to specify a matching value for the "Exchange" attribute of the underlying data but offers no way to specify the difference in opening and closing prices, the wrapper will produce a reply that accepts the predicate on "Exchange" but rejects the predicate Closing - Opening > 5. The optimizer will compensate by introducing an operator to evaluate the latter predicate at the federated server. In general, a wrapper must be able to return the value of any individual column requested in the SELECT list, but can reject any predicate or more complex SELECT list expression.

In addition to indicating that only the "Exchange" predicate will be evaluated by the data source, the reply will include a wrapper plan containing sufficient information to execute the query represented by the reply. For instance, the wrapper plan might contain a parameterized URL,
equivalent to the one a browser would produce if a user had filled out the query form manually. At execution time, the federated server will return this plan to the wrapper, which will extract the URL, submit it to the web site, parse the resulting data stream, and return the requested values to the federated server.

**Using a federated database system**

Why is a federated system useful? How do customers use federation capabilities? In general, a federated system is useful in any situation in which there are multiple sources of data, and a need to combine the information from these various sources. In this section we look at how some customers are using IBM's federated technology to solve their business problems today.

**Distributed operations: a major pharmaceutical company**

Many companies today are global companies, with a need to coordinate activities in multiple locations throughout the world. For example, a pharmaceutical company might have research labs in both Europe and the U.S. Each of the labs houses scientists looking for new drugs to battle particular diseases. The scientists all have access to databases of chemical compounds, stored in special-purpose systems that allow searching by particular characteristics of the compounds or by chemical structure (structural similarity). In both labs, the scientists run high throughput screenings of compounds to test their effectiveness against different biological targets. The results of these tests are stored in relational databases at each lab. Other data sources accessed by the scientists include large flat files of genomic and proteomic information, patent databases, spreadsheets of data and analysis, images and text documents.

The scientists in the two labs have different missions, different cures or treatments that they are pursuing. This leads them to do different experiments, and to focus on particular sets of compounds. However, often the same compounds may be useful against different targets, and sometimes one test may be a good indicator of results for other tests. Thus it is important for the scientists at one lab to have access to the data being produced at the other, so as not to duplicate effort. While this could be accomplished by building a large warehouse with all the compound data and test results, there are several drawbacks to that approach. First, the test result data changes rapidly, with thousands of records being added every day from both sides of the Atlantic, making maintenance difficult. Second, the warehouse must either be replicated at both sites, or one site must suffer slower performance for accessing the data. Replication adds to the cost of the solution and the complexity of maintenance. Third, the compound data, today stored in specialized repositories, would need to be migrated to a relational base, including re-implementing the search algorithms and any existing applications.

A federated solution eliminates these issues. Data is left in the existing data sources, with their native access paths, and current applications run unchanged. However, it is easy to build new applications that can access data from any of the sources, regardless of continent. Local data stays local, for rapid access. The less frequently used remote data is still accessible, as needed, and queries are optimized by the federated server to ensure that they are retrieved as efficiently as possible. Replication can still be used if desired for those portions of the data that are heavily accessed by both laboratories.
Lightweight extract transform load (ETL)

Many businesses choose to keep multiple copies of their data. For example, one major retailer with outlets all over the United States backs up data from its various locations at regional warehouses. The retail outlets use one relational database management system; the warehouses are implemented using another DBMS that scales better. However, this poses the problem of how to transfer the data from source to warehouse. IBM's federated technology makes it easy to not only move data, selecting it from the sources and inserting it into the warehouse, but to re-shape it as well, aggregating information from the various outlets before inserting it into the warehouse.

Distributed data warehouse

Implementing a distributed data warehouse has been shown to provide higher availability and lower overall cost. An enterprise can create several data marts that store only high level summaries of data derived from the warehouse. With IBM's federated technology, data marts and warehouse can be on separate systems, yet users of the data mart can still drill down with ease from their local level of summarization into the warehouse. Federated technology shields the users, who have no need to know that the data warehouse is distributed, by providing a virtual data warehouse.

Geospatial application

A bank needs to choose a location for a new branch office. The location chosen must maximize the expected profit. To do so, the bank needs to consider for each location the demographics of the surrounding neighborhood (Do the demographics fit the targeted customer base?), the crime rate in the area (A low crime rate is important for retail operations.), proximity of the site to major highways (to attract customers from outside the immediate area), proximity of major competitors (A site with little competition will most likely mean higher sales.), and proximity to any known problem areas that must be avoided (A dump or other unattractive feature of the neighborhood could negatively impact business.). Some of the needed information will come from the bank's own databases. Other information must be retrieved from external data stores with information on the community. This application illustrates the need to integrate geospatial data with traditional business data. It requires advanced query analysis functions for correlating the data, and end-user tools that can visually display the data in a geospatial context.

Traditionally, geospatial data have been managed by specialized geographic information systems (GISs) that cannot integrate spatial data with other business data stored in the company's RDBMS and in external data sources. DB2 Spatial Extender is the product of collaboration with an IBM partner, the Environmental Systems Research Institute (ESRI). DB2 Spatial Extender works with an IBM federated database to give customers the best of both worlds. The customer can take advantage of the geo-spatial intelligence built into the DB2 Spatial Extender combined with the vast amount of available business information from the federated system. This enables the organization to enhance its understanding of its business, leverage the value of existing data, and build sophisticated new applications, leading to business success.
Conclusions

Despite considerable attention from the research community, few commercial database management systems have addressed the problem of integrating relational and nonrelational data sources into a federation. With its federation capabilities, IBM has made significant progress toward this goal. IBM's unique federated query processing technology allows users to enjoy all the power of DB2 SQL coupled with the power of individual data sources. It provides users with all the benefits of transparency, heterogeneity, a high degree of function, autonomy for the underlying federated sources, extensibility, openness, and optimized performance. Federation is being used today to solve many important business needs.

In the future, we will continue to work to improve the performance and functionality of the federation. For example, a form of caching can already be accomplished using an automatic summary table (AST) mechanism, which allows administrators to define materialized views of data in a set of underlying tables -- or nicknames. For certain classes of queries, the database can automatically determine whether a query can be answered using an AST, without accessing the base tables. In addition to constantly improving performance, we are also working on tools to aid in the configuration, tuning and administration of federated systems. Tools for generating statistics for data from non-relational sources and for monitoring the behavior of a federated system are underway. Tools to assist wrapper developers are also in development.

Finally, even a welldesigned federated database management system and an accompanying set of tools remains a partial solution to the larger problem of data integration. A comprehensive solution will have to integrate applications as well as data, and address higherlevel issues like data quality, annotation, differences in terminology, and business rules that indicate when and how information is to be combined. IBM InfoSphere Information Server focuses on this broader set of information integration requirements to enable customers to satisfy their business integration requirements, and database-style federation is just one key integration technology.
Related topics


